Отчет по практической работе №2

Основы работы с технологиями контейнеризации и ботами Telegram

Цель работы:

Создание сервера с постоянно работающим Telegram ботом.

Ход работы:

С помощью Windows PowerShell подключаемся к серверу-шлюзу.

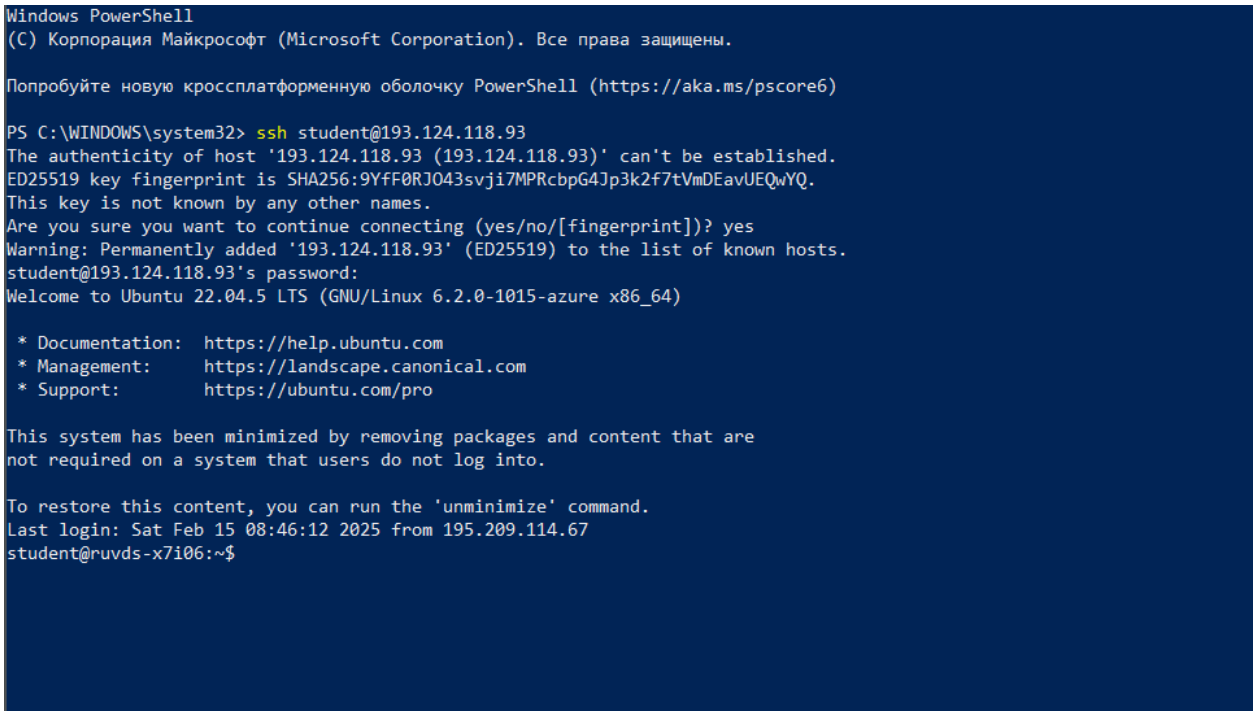

Рис 2.1

Отсюда подключаемся к рабочему серверу.

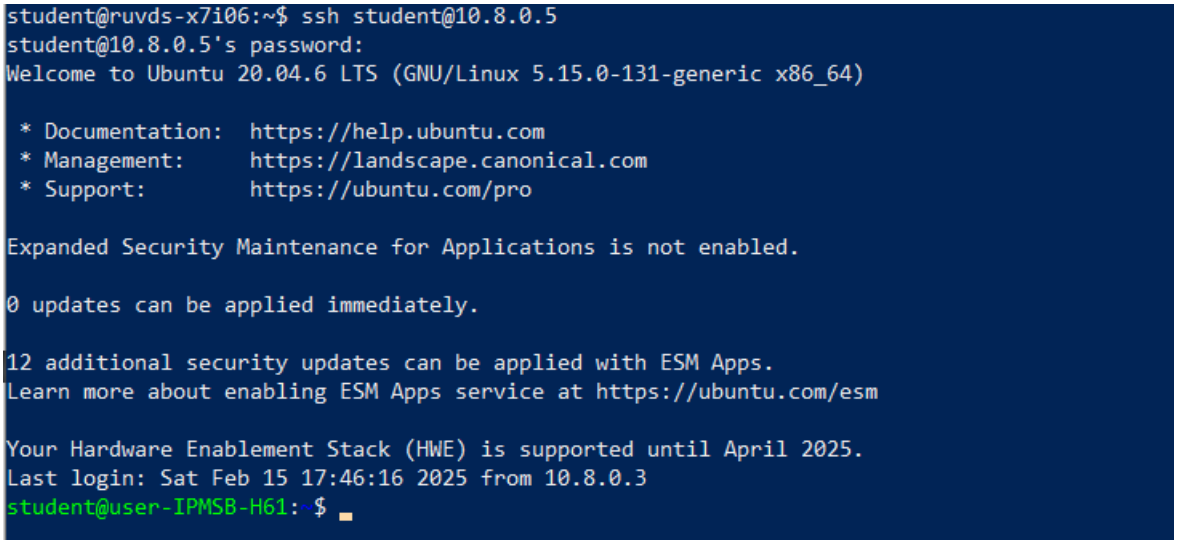

Рис 2.2

Создаем свой каталог по номеру зачетки и переходим в него и проверяем работу Python.

```
student@user-IPMSB-H61:~$ mkdir 220803144 && cd 220803144
student@user-IPMSB-H61:~/220803144$ python3.10
Python 3.10.16 (main, Dec  4 2024, 08:53:37) [GCC 9.4.0] on linux
Type "help", "copyright", "credits" or "license" for more information.
>>>
```

Рис 2.3

Создаем и активируем окружение Python.

```
student@user-IPMSB-H61:~/220803144$ python3.10 -m venv env
```

Рис 2.4

```
student@user-IPMSB-H61:~/220803144$ source env/bin/activate
(env) student@user-IPMSB-H61:~/220803144$
```

Рис 2.5

Устанавливаем telepot.

```
(env) student@user-IPMSB-H61:~/220803144$ pip install telepot==12.7
```

Рис 2.6

Создаем нового бота через BotFather.

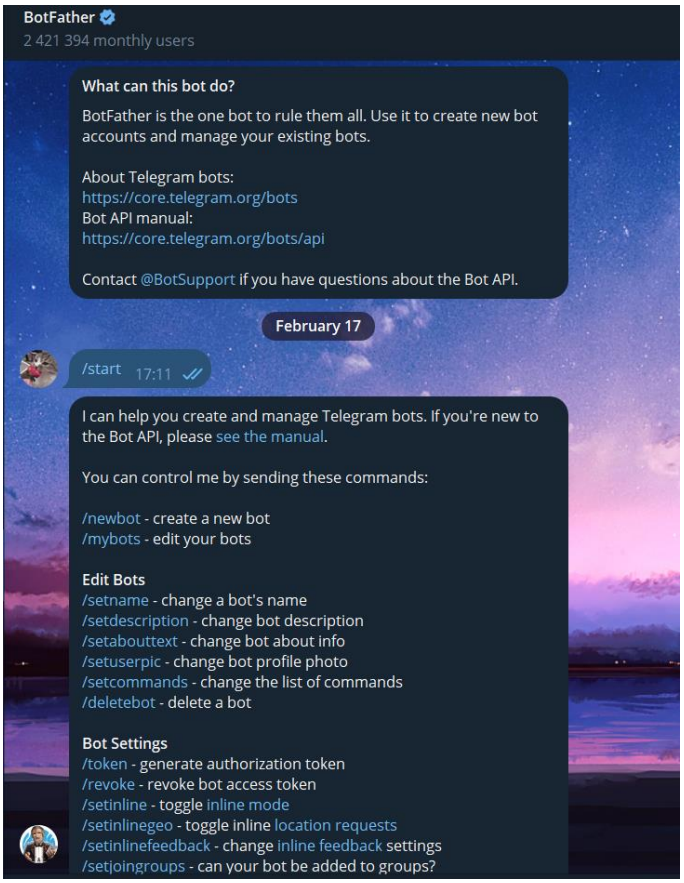

Рис 2.7

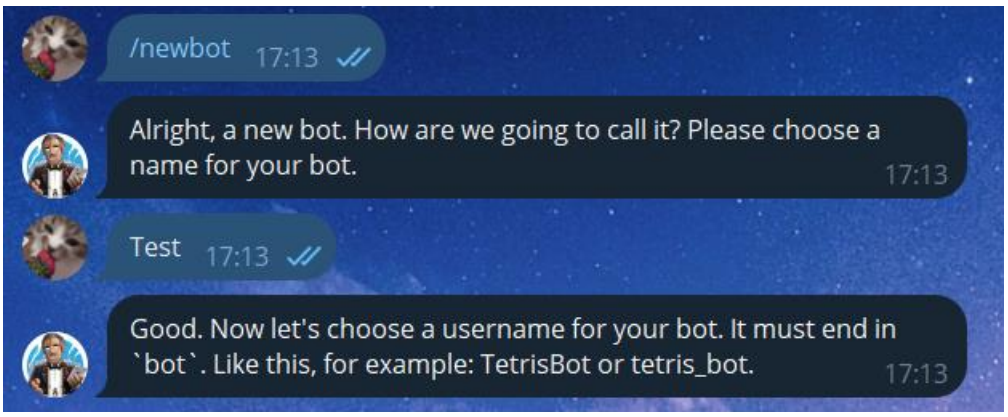

Рис 2.8

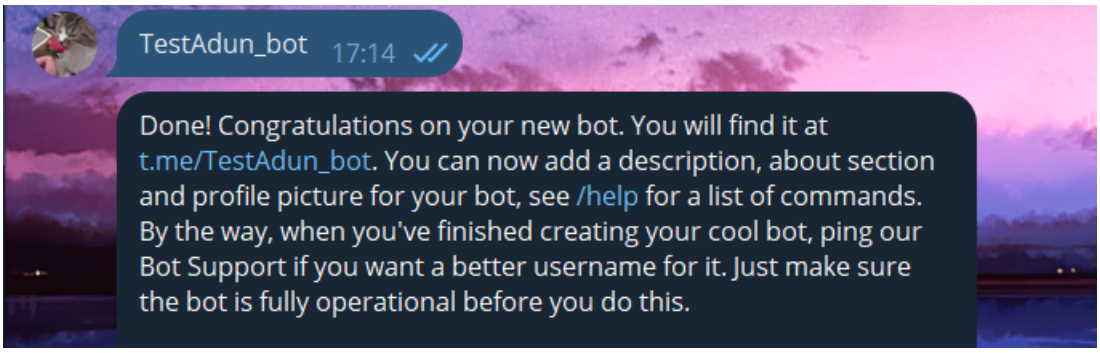

Рис 2.9

Запускаем текстовый редактор и вставляем код программы, заменив токен на свой.

```
student@user-IPMSB-H61: ~/220803144
  GNU nano 4.8                                  bot.py
import telepot
import time
def handle(msg):
 chat_id = msg['chat']['id']
 command = msg['text']
 print('Got command: %s' % command)
 print('From : %s' % chat_id)
 if command == '/command1':
 bot.sendMessage(chat_id, 'Oks')
 elif command == '/command2':
 bot.sendMessage(chat_id, 'Ok')
bot = telepot.Bot('***** PUT YOUR TOKEN HERE *****')
bot.message_loop(handle)
print('I am listening ...')
while 1:
 time.sleep(10)
```

Рис 2.10

Запускаем программу и проверяем работу бота.

```
(env) student@user-IPMSB-H61:~/220803144$ python bot.py
I am listening ...
Got command: /start
From : 1017881787
Got command: /command1
From : 1017881787
Got command: /command2
From : 1017881787
```

Рис 2.11

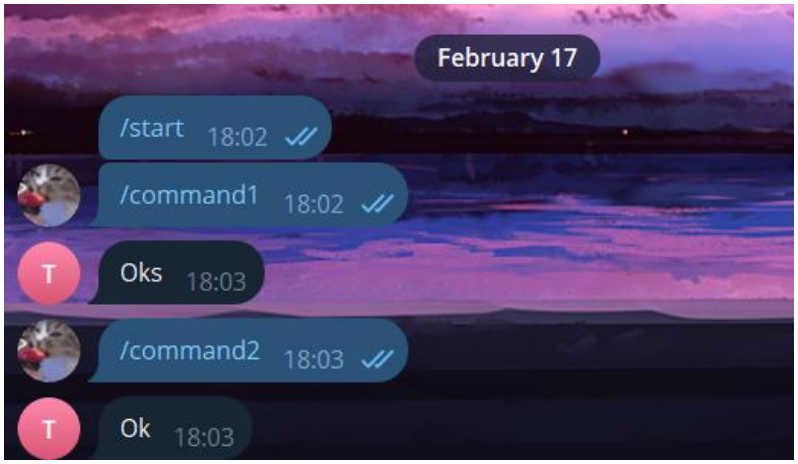

Рис 2.12

Отключаем python env и создаем файл requirements со списком библиотек, необходимых для работы программы.

```
(env) student@user-IPMSB-H61:~/220803144$ deactivate
student@user-IPMSB-H61:~/220803144$ nano requirements.txt
```

Рис 2.13

Создаем файл для сборки docker образа и собираем его.

```
student@user-IPMSB-H61:~/220803144$ nano Dockerfile
```

Рис 2.14

```
student@user-IPMSB-H61:~/220803144$ docker buildx build -t 220803144 .
[+] Building 7.8s (13/13) FINISHED                                                                    docker:default
 => [internal] load build definition from Dockerfile                                                           0.7s
 => => transferring dockerfile: 295B                                                                           0.0s
 => [internal] load metadata for docker.io/library/python:3.10                                                 2.2s
 => [internal] load metadata for docker.io/library/python:3.10-slim                                            2.1s
 => [internal] load .dockerignore                                                                              0.7s
 => => transferring context: 2B                                                                                0.0s
 => [builder 1/3] FROM docker.io/library/python:3.10@sha256:76f22e4ce53774c1f5eb0ba145edb57b908e7aa329fee75eca69b  0.0s
 => [stage-1 1/4] FROM docker.io/library/python:3.10-slim@sha256:66aad90b231f011cb80e1966e03526a7175f058672498196  0.0s
 => [internal] load build context                                                                              0.6s
 => => transferring context: 527B                                                                              0.0s
 => CACHED [stage-1 2/4] WORKDIR /code                                                                         0.0s
 => CACHED [builder 2/3] COPY requirements.txt .                                                               0.0s
 => CACHED [builder 3/3] RUN pip install --user -r requirements.txt                                            0.0s
 => CACHED [stage-1 3/4] COPY --from=builder /root/.local /root/.local                                         0.0s
 => [stage-1 4/4] COPY ./bot.py .                                                                              1.2s
 => exporting to image                                                                                         1.0s
 => => exporting layers                                                                                        0.6s
 => => writing image sha256:7ff6e5bfc23d7ed701379bf98090e6d3b10af5c5c113576a82b6d7dc6af681f6                   0.1s
 => => naming to docker.io/library/220803144                                                                   0.1s
```

Рис 2.15

Запускаем docker образ в режиме работы в фоне. Получаем CONTAINER ID.

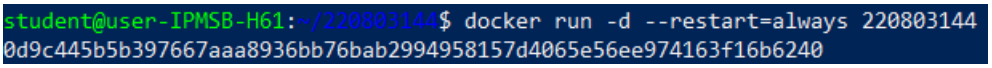

Рис 2.16

Используем CONTAINER ID для просмотра log.

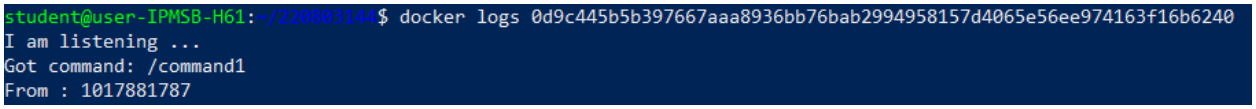

Рис 2.17

Сохраняем docker image в виде архива.

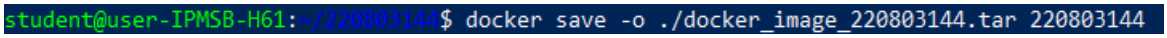

Рис 2.18

Отключаемся от рабочего сервера и копируем архив docker-образа на шлюз.

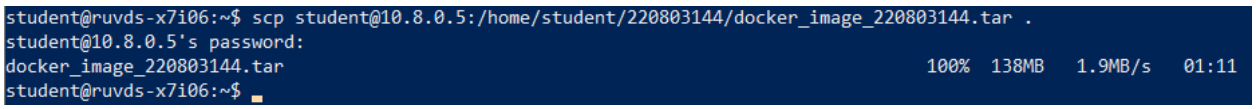

Рис 2.19

Отключаемся от шлюз-сервера и копируем файл архива к себе на ПК.

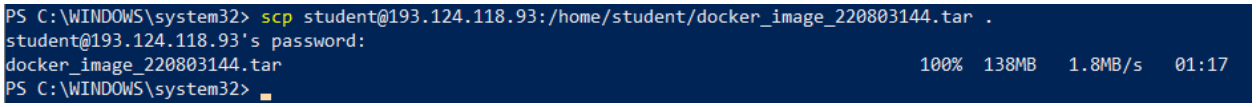

Рис 2.20

Вывод: В результате практической были изучены основы работы с технологиями контейнеризации и ботами Telegram. Был создан Telegram-бот, который отвечает на две команды, демонстрируя базовые возможности взаимодействия с пользователем. Для реализации проекта использовалась технология Docker, что позволило создать и управлять изолированными средами, обеспечивая легкость развертывания и масштабирования приложения.